4887-4002
PATENT

UNITED STATES PATENT APPLICATION

For

A METHOD AND SYSTEM FOR INTEGRATING NETWORK-BASED FUNCTIONALITY INTO PRODUCTIVITY APPLICATIONS EMPLOYING SPREADSHEETS

Express Mail label No.: EL762622154US
Date of Deposit: August 14, 2001

MORGAN & FINNEGAN, LLP
345 PARK AVENUE
NEW YORK, NEW YORK 10154-0053

A METHOD AND SYSTEM FOR INTEGRATING NETWORK-BASED FUNCTIONALITY INTO PRODUCTIVITY APPLICATIONS EMPLOYING SPREADSHEETS

Cross-Reference to Related Applications

The present application claims priority under 35 USC § 119 to U.S. Provisional Patent Application Serial No. 60/217,361, filed on July 11, 2000, for "System And Method For Integrating Network-Based Functionality Into Productivity Application" and is a continuation-in-part of U.S. Patent Application Serial 09/675,424 entitled "A Method And System For Integrating Network-Based Functionality Into PC Applications And Documents" filed on September 29, 2000, both of which are hereby incorporated by reference.

Field of the Invention

The present invention relates generally to a method and system for providing network-based functionality within productivity applications and documents, and more particularly, to a system and method for embedding network-enabling objects into spreadsheets and providing network-based functionality thereto.

Description of Related Art

There exists today a rich class of end user applications, generally termed "productivity applications," which may be used to create, view and modify personal computer (PC) documents. Examples include MICROSOFT OFFICE, MICROSOFT WORKS, COREL WORDPERFECT SUITE, and the like. These are readily distinguishable from Internet browsing applications which are used to view content on the Internet, since productivity applications have no network-based functionality, as is found in Internet browsing applications.

Readily available productivity applications include those used for (1) creating spreadsheets which manage and manipulate numerical data, (2) creating documents drawings and illustrations, and (3) creating blue prints schematics using computer-aided design (CAD) programs. Documents created and managed by such applications are typically either stored on a local hard drive of a personal computer (PC) or other computing device running the productivity application or a file server on a local network serving that PC. The content of these documents is generally self-contained. Therefore all content presented to the user by a productivity application is stored within the documents themselves.

On the other hand, there exists a vast quantity of information and variety of services that are hosted remotely and accessible by diverse users through a network. A web browser, for example, is a convenient vehicle for accessing such content and even for hosting simple applications that are downloaded from the network which work with such remote information and services. Such simple applications may manifest themselves as more advanced web pages, Java applets, browser "plug-ins", or the like. The extended functionality supported by such simple applications often attempts to recreate a segment of functionality found in an existing desktop application. However, none of these additions to web browsers (or other network clients) incorporate the full functionality of commonly available productivity applications. Moreover, as additional functionality is deployed to web browsers in attempts to raise their level of functionality, bandwidth requirements increase causing the application to become more prohibitive to download.

In general, documents managed by productivity applications are self-contained and do not allow viewing or manipulation of remote content or access to functionality provided by remote systems and servers on a network. Certain productivity application allow users to

embed static web page links or e-mail address links into a document created by the underlying productivity application. These productivity applications allow users to launch Internet clients by clicking on the link. However, it is important to note that the result of performing such an operation from within the productivity application is for the underlying productivity application to separately launch the corresponding Internet client, essentially passing the end user to a distinctly new application environment. Since the content being viewed in the Internet client, such as a web page or a new e-mail, cannot be directly displayed by the original underlying productivity application, the user is forced to manually paste the content within the productivity application in order for the productivity application to be able to manipulate or display such content. Even when such pasting is performed, any change to the underlying web page or e-mail will not be updated within the productivity application without performing further pasting. Accordingly, manipulation of network-based content within a productivity application is heretofore unknown.

Common mechanisms for deploying and utilizing network technologies include Internet clients such as web browsers, media players, and e-mail applications. Examples include INTERNET EXPLORER, MICROSOFT OUTLOOK, and NETSCAPE NAVIGATOR. Similarly, the content presented and manipulated by these various Internet clients cannot be integrated fully into the productivity application's environment such that the content can then be manipulated by the native functionality and tools resident within the productivity application. As a result, while such browsers, media players, e-mail clients, and other network technologies allow access to information on the Internet or other networks, they do not provide the functions and operations that are typically performed by a productivity application. For example, while a browser allows a user to search the World Wide Web, one cannot directly perform ad-hoc

analysis of stock data retrieved from the Internet using models running within a spreadsheet application. As a result, end users must manually move the content into a productivity application geared towards such analysis.

Attempts have also been made to receive and collect remote content from external sources from within a productivity application for use toward particular specialized tasks. For example, certain personal financial management applications retrieve stock data from the Internet and bring the data into local views to augment existing content. Unfortunately, once the information has been "imported" the local view remains static, instead of updating itself automatically as the remote content changes. Moreover, while the ability to integrate this imported static information with local content or distributed content from other sources exists in certain limited situations, no means are provided by the productivity application to extend the access capabilities to include other sources of remote content. Users are therefore limited solely to the types of information/data provided by the remote information source.

At the same time, other productivity applications collect and extract data/information from multiple sources and bring the data/information into some central document associated with a productivity application, such as spreadsheets, databases and the like. These productivity applications provide the capability to extract data/information from one or more sources and marshal the data/information into one central location. However their functionality is primitive. For example, a user may be limited to extracting particular data (such as a table of data, but not an individual row or a single paragraph) from a public or static web page, or unable to supply required credentials to gain access to a private web site. Additionally, the functionality exists solely within the application at hand (a user cannot for example apply a query tool for one application to seamlessly retrieve data for another application). These existing

tools are merely provided as limited enhancements to the existing application, but fail to provide a platform for development of a wide range of enhanced functionality for said application.

Furthermore, constructing distributed applications that manipulate remote content but execute within a network client such as a browser represents an onerous task at best. The remote content cannot be saved locally and manipulated while disconnected from the remote source and later reused when connected to the original remote location. Additionally, as mentioned above, as the level of functionality approaches that of a productivity application, the resulting browser application becomes prohibitively large for typical distribution via Internet download.

Moreover, existing desktop applications may provide a method of extending their native capabilities via a scripting interface. However these scripting environments are often inferior to typical robust development platforms. When attempting to construct solutions in these environments, developers must resort to more complex code and debugging methods, thereby additionally raising development costs and reducing the availability of developers who may have the necessary skill sets to implement such solutions. Therefore a need exists for a more robust development platform, allowing a developer or end user to quickly build and deploy applications that expose network functionality integrated with productivity application functionality.

As such, on one hand, existing productivity application fail to incorporate Internet and/or other general network-based access capabilities as an integral part thereof. On the other hand, Internet clients fail to incorporate the full scope of productivity application functionality therein. As a result, these productivity applications cannot be readily utilized for network-based operations and functionality. Furthermore, available productivity applications fail to provide

solid mechanisms for incorporating distributed and remote information into their respective documents. For example, while the ability to receive network-based information, such as stock quote streams, in real time, directly into a spreadsheet, or to receive live video feeds within a document within existing productivity applications, exists today in certain limited situations, coupling dynamic remote content with documents that are automatically updated when the remote content changes requires functionality currently unavailable in existing productivity applications. Thus, a user is denied access beyond the traditional functions provided by the standard Productivity application and/or Internet clients, and cannot create documents that leverage these network-based features.

In view of these disadvantages, there is a need for a system that allows seamless and integrated access to both local, as well as distributed (e.g. network-based), content and information. There is a further need to extend the network access and utilization capabilities of existing productivity applications. In addition, there is a need to provide a general, as well as extensible, framework for extending the network access and utilization capabilities of existing productivity applications. Furthermore, there is a need for a system to create, manage and utilize documents that contain a mix of immediate as well as distributed, content. There is further a need for managing spreadsheets so that they can behave as Internet applications themselves. There is yet a further need to enhance the packaging and delivery of documents so that they are suitable for web application deployment. Additionally, there is a need for managing access to and use of local content based on remote settings and controls.

Summary Of The Invention

The present invention overcomes the above-mentioned disadvantages found in existing technologies. One aspect of the invention provides a general system for augmenting existing productivity applications, such as spreadsheets and the like, with network-based functionality therein. Users are provided with the ability to utilize and leverage the resources of the network/Internet from within spreadsheets opened in their productivity application. For example, the augmented productivity application of the instant invention allows the user to browse the Internet, read and write emails, send and receive instant messages, perform page scraping, receive stock quotes in real time, access remote databases over the Internet, or perform other similar network-based functions from within a productivity application document, and in particular, a spreadsheet, and be able to use their productivity application functionality with both local and remote content in concert.

In one embodiment of the present invention, a spreadsheet with network-based functionality is disclosed. The spreadsheet is used with the productivity application augmented with network-functionality software that is resident in computer memory. The spreadsheet is coupled with network-enabling objects embedded in one or more cells that are configured to provide network-based functionality to the spreadsheet.

The invention described herein supports integration of a wide range of network-based user interfaces, content, data, and functionality into productivity applications utilizing spreadsheets. The networks that can be used in conjunction with this technology include the Internet, as well as other networks.

The invention described herein enables the development, deployment, and use of these network-based services and documents with network-based functionality. The invention allows integration of a wide range of network-based services, including Internet-based, user

interfaces, content, data, and functionality into the augmented productivity application and also into spreadsheets used or produced by such augmented productivity applications. Thus, the invention enables professional developers as well as end users to develop productivity application-based, network-enabled documents and applications by using augmented versions of productivity applications with which they are already familiar. This simplifies and improves the development of network-based applications or documents.

The present invention also provides a client/server architecture wherein one or more users may connect to a network, comprising one or more independent servers, for receiving network-based functionality within their spreadsheet productivity application.

The present invention may include an architecture that involves four software components. The first component is a document-independent, run-time client-side code, through which a productivity application is extended by network-enabling software that facilitates access to network-enhanced services and functionality from within the application environment. The second component of the architecture is the enhanced spreadsheet, which includes network-enabling objects. The third component of the architecture is a design tool for creating the enhanced personal computer document. The fourth component of the architecture is server-side code that manages interactions between clients and Internet resources and interactions among multiple end users.

These aspects and other objects, features, and advantages of the present invention are described in the following Detailed Description which is to be read in conjunction with the accompanying drawings.

Brief Description Of The Drawings

FIG. 1A is a block diagram of an exemplary embodiment of a spreadsheet in a productivity application augmented by a productivity application extender, wherein the spreadsheet may contain network-enabled objects;

FIG. 1B is a block diagram of exemplary embodiment of a spreadsheet in a productivity application augmented by a productivity application extender in cooperation with a component object model (COM) interface code;

FIG. 1C is a schematic diagram of exemplary software modules supporting the network-enabling object in accordance with the present invention;

FIG. 2 is a schematic diagram of a productivity application extended by a client extender in accordance with one embodiment of the present invention;

FIG. 3A is a schematic diagram of one embodiment of network-enabling software for use by software developers;

FIG. 3B is a schematic diagram of one embodiment of network-enabling software for use by users;

FIG. 4 is a schematic flow diagram depicting a process for providing network-enabling software to a productivity application;

FIG. 5 is a block diagram of a productivity application that provides users with built-in tools for creating documents with network-based functionality;

FIG. 6 is a schematic diagram of a spreadsheet incorporating network functionality in a spreadsheet document in accordance with the present invention;

FIGS. 7A-7D are schematic diagrams of different states of communication between the network-enabling objects in the spreadsheet with one or more controls installed on a device running a productivity application;

FIG. 8 is a schematic flow diagram depicting the process for creating a spreadsheet with network-enabling objects embedded therein;

Fig. 9 is a schematic flow diagram depicting the process for launching network-based functionality with a spreadsheet;

FIG. 10 is a schematic block diagram depicting the client/server system of one embodiment of the present invention for providing network-based functionality with a spreadsheet;

FIG. 11A-11B are screen-shots depicting two embodiments of a spreadsheet in accordance with the present invention;

FIG. 12 is a schematic block diagram depicting the client/server system of an embodiment of the present invention for developing and accessing a web page as an embedded object that is produced using a spreadsheet.

FIG. 13 is a schematic flow diagram depicting the process for developing and accessing a web page as an embedded object that is produced using a spreadsheet based productivity application;

FIG. 14 is a schematic flow diagram depicting the process of the operation by a user to use a web page created in accordance with the process of FIG. 13;

FIGS. 15A-15B are a schematic flow diagram illustrating a process for selecting a query and assigning results to a location in a spreadsheet according to one embodiment of the present invention; and

FIGS. 16A-16M are screen shots of exemplary dialog boxes presented during the process depicted in FIGS. 15A-15B.

With reference to the following detailed description, the aforementioned drawings

will be described in greater detail below.

Detailed Description of the Invention

The present invention relates to a system and method for providing network-based functionality to a spreadsheet within a spreadsheet productivity application. Initially, a productivity application, which is a software program used for a particular desired task, such as to create a functional spreadsheet document, is augmented with network-enabling software that allows users to provide new features to the productivity application. The network-enabling software of the present invention enables users to generate, modify and utilize spreadsheets with embedded network objects. Such objects can work in conjunction with the network-enabling software to launch and utilize network-based functionality from within the spreadsheet. The present invention enables the use of a wide array of public and private network-based content and services including, but not limited to, content and web services and other Internet-based content and services, within a productivity application, such that these services can be viewed and used within such applications, and within the spreadsheet associated with these applications.

It should be noted that the present invention, even when described in connection with the Internet specifically, may be applied equally to local area networks (LANs), private "Intranets," hybrid public-private "Extranets," virtual private networks (VPN), and other networks.

The services that can be provided and used within a productivity application include web browsing, electronic mail (e-mail), instant messaging, digital rights management (DRM), remote database access over the Internet, and other Internet-based services, as well as combinations of these content and services. These services may be provided from within a

spreadsheet document generated by a productivity application such as MICROSOFT EXCEL, MICROSOFT WORKS, LOTUS 1-2-3, QUATTRO PRO, and other like productivity applications.

According to an embodiment the invention, once the network-enabling software has been installed on a personal computing device, the network-enabling software may automatically load when the augmented productivity application is loaded. Users of the augmented productivity application may load existing spreadsheets or create new ones with the augmented productivity application. Once a spreadsheet is open within the augmented productivity application, the user is able to make any of the standard operations thereto, such as creating, adding or modifying content therein. Furthermore, the augmented productivity application allows the user to embed network-enabling objects in the spreadsheet by a plurality of mechanisms.

According to the invention, the network-enabling objects embedded in a spreadsheet allow users to launch network-based operations from within the spreadsheet when the associated productivity application has the network-functionality software running alongside. Such network-enabling objects may allow users to perform web browsing, electronic mail (e-mail), instant messaging, remote database access over the Internet, DRM, and/or a plurality of other network-based operations consistent with the purpose of the present invention from within spreadsheets. The invention also enables integration of productivity application as well as spreadsheets with remote applications, databases, and data sources, where communication between them may occur over a plurality of network types.

According to the invention, both fixed and customizable objects may be placed within individual spreadsheets to provide network-based functionality thereto. The network-

enabling objects can either be visible to the user within the document or hidden. The network-enabling objects may contain data, UI elements, and code such as Visual Basic scripts. According to the invention, the network-enabling objects can present the user with specific options, and act in response to user-provided information and choices, and/or they may take network-related actions automatically, driven solely by software commands.

As noted above, the actions taken by the network-enabling objects placed within spreadsheets can include any combination or sequence of a wide array of Internet-related actions, including but not limited to checking or sending e-mail, checking or sending Internet-based voicemail, querying remote databases over the Internet, requesting or updating a web page, invoking a remote application over the Internet, establishing a real-time data feed with a specific Internet data source, or sending a real-time message.

It should be noted that the ability to make use of these services and functionality is not restricted to existing productivity application and spreadsheets. Such services and functionality can also be provided through future productivity applications not yet developed, future spreadsheets not yet developed, and through augmented or enhanced future versions of existing applications and document types, or through wireless, laptop, handheld, or other computing devices.

According to another embodiment, the spreadsheet may be provided with an installation facility to install the network-enabling software on a computing device where the spreadsheet resides but where the underlying productivity application has not been provided with the network-enabling software of the present invention. The invention discloses a system for automatic installation of the network-enabling software so that new spreadsheets with network-enabling objects may be created, manipulated, enhanced and supported.

The invention further discloses the packaging of network-based functionality via software development kits (SDKs) and application programming interfaces (APIs) usable by professional productivity application developers. It will be appreciated that the functionality provided by the invention, the architecture in which they are embedded, and the APIs through which they can be utilized, greatly simplify and improve the development of new, network-enabled productivity application and spreadsheets. This functionality can also be provided for network-specific documents such as e-mail messages that are processed by SMTP/POP e-mail clients.

In general, the present invention may include an architecture that involves four software components. The first component is a document-independent, run-time client-side code, through which a personal computer application is extended by network-enabling software that facilitates access to network-enhanced services and functionality from within the application environment. The second component of the architecture is the enhanced personal computer document, which includes network-enabling objects. The third component of the architecture is a design tool for creating the enhanced personal computer document. The fourth component of the architecture is server-side code that manages interactions between clients and Internet resources and interactions among multiple users.

The runtime client-side code may be configured as one or more distinct sub-components. For example, one sub-component may constitute generic client-side code that handles aspects of system operation common to all applications, while a second sub-component may be application specific (e.g., the portion of the code that controls the application's unique user interface). This runtime support code at the client can be developed using any of a number of programming languages, such as C, C++, Java and Visual Basic. One of the functions of the

runtime client-side code is to wrap and package low-level transport protocols such as Transmission Control Protocol/Internet Protocol (TCP/IP) and hyper-text transfer protocol (HTTP) into elements that can be manipulated and that provide specific, high-level functional capabilities, such as wrapping Simple Mail Transfer Protocol SMTP to provide easy-to-use email capabilities, wrapping HTTP Web page access and parsing to allow easy access to live information on web pages. The wrapping is done in a layered fashion so that users can create and/or manipulate the elements without having any programming capabilities whatsoever, yet developers can leverage scripting or lower-level programming to access more advanced facilities.

The second functionality of this runtime client-side code is to integrate with the hosting productivity application. This integration falls generally into three categories. The first category simply provides a seamless runtime environment so that these services run and are available from within the application process. This makes the packaged services available for calling from within scripts in enhanced documents and from application code itself, and is generally enabled by running at least part of the support code as an add-in to the productivity application itself. The second category of integration is at the user interface (UI) level, where the enhanced functionality elements are exposed as operations on toolbars and menus, and as objects in enhanced documents themselves. This integration is enabled through a combination of standard windowing facilities on the client platform, and use of the application and document object models to allow integrated rendering of the UI elements and handling of associated events. This, for example, enables an end-user to drag an email button that has been provided on a toolbar into a document, such that the button is rendered as part of the document and that when the button is pressed a pre-defined message is emailed to the author of the document or other

specified recipient. The end-user would need no programming ability to perform this embedding and configuration, having only to fill in the content of the message and the name or email address of the recipient. The third integration category relates to embedded applications, and involves controlling the behavior of the hosting application itself when an embedded application has been placed in "run mode". Document-based applications are generally not designed to behave as application containers for embedded applications, and when used in this manner must be managed carefully to prevent unwarranted operations by users. This management or control can involve blocking certain operations, prevention of the embedded application from being rearranged or reconfigured, or modifying and/or augmenting various ordinary tasks such as saving a local copy of the embedded application enhanced document. This control is achieved again by packaging some or all of the runtime support code as an extension (such as an add-in or a plug-in) so that it runs in close cooperation with the hosting productivity application, and in working with the productivity application extension to change the productivity application behavior. The productivity application extension can change the productivity application behavior by a combination of handling the entry points and events, overriding handling by the productivity application itself, and extending the productivity application object model through new, custom code.

In this runtime environment, an augmented productivity application may allow an end-user to download and open spreadsheets within the productivity application's native environment. In one embodiment, the document may constitute a web page, or a productivity application version thereof, downloaded from a web server. The augmented desktop application and enhanced document together form an Internet or network-enabled application that facilitates

access to remote services and functionality while retaining functionality inherently provided by the desktop application. Access to distributed services may be via any network or protocol.

The enhanced spreadsheet includes Internet-enabling objects which can be document specific and placed into a document such as a productivity application-based web page and the like by either a developer or end-user. The UI of such objects can include buttons, lists, menus, and forms. Their functionality can include combinations of Internet or network-based functions such as sending and receiving data, sending and receiving e-mail messages, causing documents to be posted to web sites, and other functions enabled by the technology. The functionality of such objects is specified by custom-developed software, which can be associated with the object's UI, and then activated by software events and/or user-issued commands. Once written, these objects can be separately transmitted via the Internet and stored on client machines for use by developers and users.

In certain embodiments of the present invention, enhanced spreadsheets created by a developer can be defined as embedded applications, consisting of three defined elements, namely a wireframe, population data and form data.

The wireframe includes static content and scripting code that is generally not modified during runtime within a productivity application residing on an user's system. The static content also includes the layout configuration of the spreadsheet viewed by a user, such as fonts and/or colors. The script code allows users or developers to extend the built-in functionality of the productivity application or simply create macros containing often-used packaged sequences of operations (for example, repeatedly creating the same customized chart from a set of data). In the context of the wireframe, the script code may contain code to assist in supporting the present invention.

The population data is dynamically generated data used to populate specified areas within a wireframe during a desktop application's runtime. Typically, such data is supplied by a server and is designated "read only", but where appropriate the data may be editable, or can be pushed back to the server for other custom processing. Population data may be used in a variety of ways, including to populate fields, strings, and pick lists for display to the user, or transparently as calculation data operated on by the application. It may also be defined to be a function of information maintained in the associated wireframe, user input, or both. It may typically be generated or retrieved upon actions such as startup, connection, or form submission, or a manual refresh request or in real-time in response to remote data changes.

The form data is data that is specified or manipulated at the client by users during runtime. Form data is usually one-way data transmitted from client to server. But in some circumstances it may be desirable to read such data from a server (for example, as cached prior information to pre-populate a user form or as smart defaults for a user form) to pre-fill forms within a wireframe. Form data includes data items, such as data entered into text box prompts (for example, asking for a user's username and password for authentication) and parameterized data entered into defined queries (such as stock ticker symbol as a parameter in a web query).

The design tool for creating enhanced spreadsheet augments a productivity application so that a software developer is provided with functionality that is similar to a special-purpose productivity application that is oriented towards the design of complex enhanced spreadsheets, services and applications. In the design-time environment, an augmented productivity application can provide a developer with tools for creating the distributed documents described above.

It should be noted that support for the design-time environment is similar to the packaging and functionality described above for the general runtime support except that the integration with and control of the containing application is generally more extensive. In addition, the elements and tools that are exposed to a developer are generally more detailed and in many cases will involve scripting or code to achieve the greatest level of functionality and control. The principal difference in functionality in the developer's design time tools and environment from an end-user's authoring environment is that the former provides extensive controls targeted at defining and managing the behavior of the containing application, while the latter is focused on creating and managing elements in enhanced documents and/or directly accessing the distributed functionality and services of the runtime itself. The design-time support code will generally be packaged as one or more separate add-ins and/or tools from the standard runtime code. Furthermore, the runtime code can be divided along functional boundaries into multiple, separately distributed pieces as well. For example, this runtime could be sub-divided into components or libraries that are completely independent of any particular enabled application (perhaps as a collection of COM objects), an application-specific add-in that provides base end-user tools and support for enhanced documents, and a second application-specific add-in that provides for application and environment control for embedded applications, where the enhanced document is actually intended to be an embedded application and have a distinct design mode and run mode. The development environment may then be packaged as one or more additional add-ins distinct from the above.

The design tool allows the developer to design the distributed Spreadsheets described above. The developer may be provided with tools that allow him or her to embed Internet or network-related functionality (e.g., automated e-mail notifications) in the spreadsheet,

control the UI that will be presented to the end-user when the document is opened, define population-data sources and how the data is displayed, define business rules, and various other functions within the scope of the present invention.

The developer may also create parallel browser-based views of these distributed documents. The choice of whether to download a browser-based view or desktop-application-based view of a page may be controlled by preferences set by the user and stored at a server. The developer may also wish to augment the browser-based version of the page so that it provides some of the additional functionality included in the productivity application based version. Examples of such network-functionality may include scripting for form-data posting, HTML-based lightweight grid handling, and inserting meta-tags into the browser-based view of a page for managing later wireframe synchronization with the desktop application version.

Since some developers may wish to augment the browser-based version of the web application still further, the invention provides the ability to retain such modifications and enhancements through subsequent desktop application-to-augmented-HTML exports, and push modifications back to the desktop application version where appropriate. A typical example of HTML-side augmentation is inserting client-side scripting to add functionality to the UI beyond that provided by a browser (for example, ornate hover buttons). Additional desktop application-side augmentation might involve a business analyst defining macros and calculations linking various data items within the application.

The invention provides for server-side components that manage interactions between clients and Internet resources and/or interactions among multiple users, the latter greatly facilitating client peer-to-peer operations. Specific services can be characterized by the kind of server support they require. For example, E-mail notifications require basic SMTP mail services.

Semi-standardized services can be leveraged for functionality such as remote database access, messaging, conferencing and the like. Custom remote services can be leveraged for accessing proprietary information feeds (such as real-time stock quotes), document archives and repositories, corporate workflow systems and the like. For these latter categories, the present invention adds value by providing a generalized framework for finding and accessing the remote services, and for opening standardized connections for communication and data transfer to the client productivity application.

With reference to the figures, various embodiments of the present invention will now be described in greater detail. FIG. 1A provides an illustration of a system in accordance with the present invention. A productivity application 100 is provided with a spreadsheet 110. In accordance with the present invention, the productivity application 100 is not limited to any particular task, form or vendor. According to the present invention, the computing device is not limited to any particular type of personal computer, and is operable on any machine with a suitable operating system having graphical user interface (GUI) facilities. The operating system for use in the present invention is also not limited to any particular type, and may include MICROSOFT® WINDOWS®, MACINTOSH, UNIX, LINUX, NET BSD, FREEBSD, and the like. This functionality can also be provided to applications running on Personal Digital Assistant (PDA) devices, cellular phones and other "light clients" connected to the Internet via wireless communications.

According to the invention, the productivity application 100 is augmented so that it is capable of developing, using, accessing, manipulating, and displaying network-enhanced documents. Such an augmented productivity application 100, which allows for the creation and use of network-enhanced documents, can essentially be termed an application within an

application. The present invention enables access to Internet-based services, data, and functionality within network-enhanced documents, which are developed, edited, and viewed via the augmented productivity application 100. The invention provides for tools, pre-defined scripts, and supporting runtime code that make construction of such enhanced documents in the desktop application's native format simple enough for users, yet flexible and comprehensive enough for developers to rapidly construct complex, powerful applications that leverage both Internet/network services and the client-side functionality of the particular productivity application. Where possible, the technology leverages a built-in script execution capability provided by the productivity application 100, such as Visual Basic for Applications facilities provided by MICROSOFT. Thus, this system provides a powerful environment for developing and using enhanced documents with rich and familiar client-side functionality.

The spreadsheet 110 is an object that is typically created, used, manipulated, augmented and saved by the underlying productivity application 100. For example, a spreadsheet is the document where the underlying productivity application 100 may be a spreadsheet application, such as MICROSOFT® EXCEL or COREL® QUATTRO®. The spreadsheet 110 may be provided with embedded or associated code that provides access to distributed, usually Internet-based, services, data and functionality residing on multiple computer systems connected via the Internet or other network. These services and functionality may be made available via the Internet using the typical network protocols, such as HTTP ___ transfer protocol, FTP, SMTP (e-mail), or streaming media.

Productivity application 100 is provided with a productivity application extender 120. The productivity application extender 120 enables users to augment the productivity application 100 with network-based functionality. The spreadsheet 110 is provided with

network-enabling objects 130 that provide network-based functionality to the spreadsheet 110. The network-enabling objects 130 may be used to provide any network-based functionality from within the spreadsheet 110. For example, the network-enabling objects 130 may be used for browsing the Internet, screen scraping, sending and receiving electronic mail (e-mail), instant messaging, chat, internet telephony, streaming media, stock quote retrieval, web page update detection, workflow, conferencing, licensing, automatic notification of a certain event, usage monitoring, replication and synchronization and other similar network-based tasks and content.

The network-enabling object 130, which is placed inside the spreadsheet 110, can perform a wide range of network-based functions, as specified by a developer. Such network-enabling objects 130 can include UI elements, such as buttons, pick lists, and menus, program code, such as Visual Basic scripts that perform functions, and data elements. These objects can be placed into the spreadsheet 110 by developers and users, where they remain as part of the spreadsheet 110 structure, upon saving the spreadsheet 110.

Coupling the productivity application extender 120 to the embedded network-enabling object 130 provides one or multiple network-based capabilities, such as web browsing and e-mail within the productivity application 100. As a result, the productivity application 100 serves as the UI to the Internet and other networks, which eliminates the need to rely on separate Internet-specific client applications, such as web browsers (e.g., NETSCAPE NAVIGATOR), e-mail clients (e.g., MICROSOFT OUTLOOK or EUDORA), and media players (e.g., distributed by REAL NETWORK). Furthermore, the provision of web access within the productivity application 100, employing the native data types and facilities of the productivity application 100 for the representation of web pages, allows users to use the web site from within the productivity application 100. This allows users to have access to functions such as browsing pages,

navigating through hyperlinks or image maps, entering data in forms, and a variety of other network-based functions.

According to one embodiment of the invention, the productivity application extender 120 may interact with one or more Component Object Model (COM) components on a local computing device to support and provide additional functionality within the productivity application 100 as well as spreadsheet 110, as shown in FIG. 1B. COM is a software architecture that allows components made by different software vendors to be combined into a variety of applications. COM defines a standard for component interoperability and provides the underlying support for Object Linking and Embedding (OLE) items and ActiveX controls to communicate with other OLE objects or ActiveX controls. OLE is a framework for a compound document technology, available from MICROSOFT. OLE is a set of APIs used to create and display a compound document. An ActiveX control is a software module based on the COM architecture, which enables a program to add functionality by calling ready-made components that blend in and appear as normal parts of the productivity application. ActiveX controls are typically used to add user interface functions, such as 3-D toolbars, a notepad, calculator or even a spreadsheet. In many cases, COM provides the underlying services of interface negotiation, life cycle management (determining when an object can be removed from a system), software licensing, and event services (putting one object into service as the result of an event that has happened to another object).

According to the present invention, the network-enabling objects 130 communicate with the productivity application extender 120 to enable the document to receive network-based functionality therein. The productivity application extender 120 communicates

with the COM component(s) 140 to allow the embedded network-enabling objects 130 in the Spreadsheet 110 to provide additional functionality to the spreadsheet 110.

FIG. 1C provides an illustration of one embodiment of the network-enabling object 130 in accordance with the present invention. The network-enabling object 130 is used to launch the network-based functionality within the spreadsheet 110. The network-enabling object 130 is embedded in the spreadsheet 110 so that the functionality of the network-enabling object 130 becomes at integral part of the spreadsheet 110. The network-enabling object 130 comprises two main components, initialization code 190 for initializing and launching the network-based functionality prior to runtime, and runtime code 192 for providing the network-based functionality during runtime. As a result, the initialization code 190 causes the launching of the network-based functionality, while the runtime code 192 enables a user to continuously receive network-based functionality within the spreadsheet 110.

It should be noted that the initialization code 190 plays no role during runtime, and remains static/dormant during runtime of the network-based functionality. According to one embodiment, the runtime code 192 can be provided with all the necessary functions or routines for enabling the productivity application extender 120 to read the necessary information and then initialize and launch the network-based functionality. According to another embodiment, the spreadsheet 110 can be embedded with code that includes some of the necessary functions/routines to initialize and launch the network-based functionality, and the initialization code 190 can be provided with the bare minimum routines to merely initialize the network-enabling object 130.

According to one embodiment, the network-enabling object 130 may be provided with a UI 232 that makes the network-enabling object 130 visible to the user. It should be noted

that providing the UI 232 for the network-enabling object 130 is strictly optional, since a number of network-enabling objects 130 may be provided such that they are hidden from the user's view while still providing network-based functionality.

FIG. 2 provides an illustration of a productivity application 100 in association with the network-enabling software 210 of the present invention and other generic add-in(s) 220 that may be supplied by one or more third-party vendors or developers. As noted above, the productivity application 100 is not limited to any particular task, form or vendor. The productivity application 100 may be a spreadsheet application. Furthermore, the productivity application 100 may be provided by any vendor, including MICROSOFT, COREL, SUN MICROSYSTEMS and the like.

The productivity application 100 comprises native productivity application code 230. The native productivity application code 230 includes code that makes the productivity application 100 operational and provides the functionality and features that are standard for the productivity application 100. The native productivity application code 230 generally comprises the code that creates and provides the user interface (UI) 232 for the underlying productivity application 100.

The native productivity application code 230 exposes a native object model 235. The native object model 235 provides a description of an object architecture, including the details of the object structure, interfaces between objects and other object-oriented features and functions. The native object model 235 enables the use of add-ins 220 and network-enabling software 210 so that the functionality of the add-ins 220 and network-enabling software 210 can seamlessly be integrated with the original functionality and features of the productivity application 100.

The native productivity application code 230 may also include an Application Program Interface (API) 240, which may be a part of the native object model 235. The API 240 provides a language and message format used by the productivity application 100 to communicate with the operating system or some other system or control program or communications protocol. The API 240 is implemented by custom-authored components, which leverage the API 240 and manipulate the productivity application 100. The API 240 also provides a method for extending the productivity application 100 by the productivity application extender 130 or other COM objects, allowing software developers to author custom functionality; as a result, the API 240 allows manipulation of the operation of the productivity application 100 from an external source, as opposed to from within the productivity application 100 and/or the spreadsheet 110. The API 240 provides developers a blueprint of how to manipulate the productivity application 100 by publishing a consistent set of interfaces.

The native productivity application code 230 provides a native script engine 250. The native script engine 250 is a facility in the productivity application 100 that can execute interpreted languages, such as Visual Basic, JScript scripts and/or other types of scripts. The native script engine 250 allows the script code in the spreadsheet 110 to execute and manipulate the productivity application 100 environment.

The native productivity application code 230 may include a productivity application extension enabler 260 that allows and facilitates the use of add-ins and/or plug-ins with the productivity application 100. The productivity application extension enabler 260 enables add-ins and/or plug-ins that have been added to the productivity application productivity application 100 to be loaded and run automatically when the productivity application 100 is launched. Generally, the productivity application extension enabler 260 is a manifestation of the

API 240. As such, the productivity application 100 may either have an API 240 or a productivity application extension enabler 260.

Using the available features of the productivity application 100, a user can create a new spreadsheet 110 or open an existing spreadsheet 110 to perform operations thereon. The network-enabling software 210 allows users to embed network-enabling objects into the spreadsheet 110, which will be described in greater detail below. Once the network-enabling objects are embedded in the document 110, the network-enabling software 210 allows a user to use the network-enabling objects and receive the network-based functionality in their documents 110, as well as use the standard functions and features provided by the productivity application 100.

FIG. 3A provides an illustration of one embodiment of the network-enabling software 210. The network-enabling software 210 includes the productivity application extender 120, which provides any standard productivity application 100 with one or more network-based capabilities.

The productivity application extender 120, which provides runtime support, includes network-functionality services 310 as well as application services 315. The network-functionality services 310 may include session management, security and location management that provide network-based functionality, as well as some combination of the individual network-functionality services. The network-functionality services 310 provide runtime support for the network-based functionality. The network-functionality services 310 expose an API with methods than can be called by the network-enabling objects 130 and scripts from within the Spreadsheet 110 to use one or a plurality of network-based functionality. The network-functionality services 310 also includes the base functionality for packing away script code of the

spreadsheet 110 for safe and easy network transport as well as unpacking a script code 720 (FIG. 6) of the Spreadsheet 110 when the Spreadsheet 110 is opened.

The application services 315 may include XML parsing and other handling, event handling and service management, query routing, document packaging, caching, data initialization and persistence, command routing and other similar services, as well as some combination of the individual application services. The application services 315 provide runtime support for the embedded applications and/or network-enabling objects 130 in the Spreadsheet 110.

According to another embodiment, the productivity application extender 120 is a COM component. The productivity application extender 120 is deployed as a dynamic link library (DLL) file, which is not launched directly by the user but may be called by the productivity application 100 for receiving network-based functionality. As such, the productivity application extender 120 may also be used by any application running system-wide, in addition to being accessible within the productivity application 100. It should be noted that designing the productivity application extender 120 as a DLL file is just one method of deployment, and various other manners of deployment exist within the scope of the present invention. As such, each functionality of the productivity application extender 120 may be provided as a separate routine, class or function in the DLL file, which may be packaged to include one or more of the network-based functionality listed above.

It should be noted that the productivity application extender 120 may be provided with just the network-functionality services 310 or just the application services 315 or a combination of the two. Furthermore, the scope of the present invention is not limited by the

types of services included in the network-functionality services 310 and/or the application services 315.

The network-enabling software 210 may further be provided with developer tools 320, which provide a plurality of functionality to a developer who wishes to create an enhanced spreadsheet 110. The developer tools 320 may automatically be invoked upon the launching of the productivity application 100. According to one embodiment, the developer tools 320 have a UI for display to the developer in the form of a toolbar. The developer tools 320 also allow manipulation of an information model 790 (FIG. 7D) for the productivity application 100, including handling of wireframe data, population data, and the queries used to retrieve the population data. The developer tools 320 provide software developers with the ability to construct network-enabled productivity application and create end-user tools and utilities, and network-enabling objects 130. The developed functionality may be used for various network-based capabilities. It should be noted that the functionality is developed in accordance with the information model 790 (FIG. 7D).

The developer tools 320 are configured to provide various sets of functionality, enabling a developer to build and deploy a compatible network-based spreadsheet. The functionality may include layout utilities, forms management, query management, data modeler, a packaging assistant, and other miscellaneous utilities and operations that assist in network-based activities, along with any combination of one or more of the enumerated functions.

According to one embodiment, the developer tools 320 are created and packaged as a COM component. The developer tools 320 package is deployed as a DLL file. As such, the developer tools 320 may also be used by any application running system-wide, in addition to being accessible within the underlying productivity application 100. According to another

embodiment, each functionality of the developer tools 320 may be provided as a distinct COM component that can be installed by the software developers at their own wish. According to yet another embodiment, groups of functionality of the developer tools 320 may combined into single COM components that can be installed by software developers at their own wish. It should be noted that designing the developer tools 320 as a DLL file is just one method of deployment, and various other manners of deployment exist within the scope of the present invention.

The productivity application extender 120 and the developer tools 320 are integrated or coupled to the productivity application 100 through the productivity application extension enabler 260. This integration may be achieved through standard methods of extensibility. It should be noted that the productivity application 100 may make itself extensible through a variety of means. For example, Microsoft's products can be extended through what they term as an "add-in" model, where the COM object extenders, such as the developer tools 320 and the productivity application extender 120, are defined as add-ins at design time, through, for example, the use of a wizard or by implementing a required programming interface.

FIG. 3B provides an illustration of a second embodiment of the network-enabling software 210 for use by users. The network-enabling software 210 is provided with the productivity application extender 120 that provides the productivity application 100 with a plurality of network-based capabilities, as well as a plurality of user tools 330.

As noted above, the productivity application extender 120, which provides runtime support, includes network-functionality services 310 as well as application services 315. It should be noted that the productivity application extender 120 may be provided with just the network-functionality services 310 or just the application services 315 or a combination of the

two. Furthermore, the scope of the present invention is not limited by the types of services included in the network-functionality services 310 and/or the application services 315.

The network-enabling software 210 may further be provided with a plurality of user tools 330. The user tools 330 provide the ability for a user to receive network-based functionality within the desired productivity application 100. The user tools 330 enable users to embed network-enabling objects 130 and functionality into their Spreadsheets 110 through the use of toolbars, user wizards and the like. In other words, the user tools 330 may comprise a UI that blends in with the UI 232 of the productivity application 100, wherein the users can utilize the UI of the user tools 330 to seamlessly embed the desired objects 130 for the network-based functionality in the spreadsheet 110.

According to one embodiment of the invention, the user tools 330 are configured to provide various optional features, such as web page scraping, embedded tables that might support, for example, stock quotes in real time, and other miscellaneous utilities and operations that assist in network-based activities within the spirit of the present invention. As a result, the user tools 330 enable users to embed live data feeds, create objects to send and receive e-mails, selected web pages to embed in the spreadsheet 110, and other similar network-based functions. The user tools 330 allow embedding of network-enabling objects 130 that use the network-functionality services 310 at runtime to achieve the network-based functionality.

As noted above, the user tools 330 often work in conjunction with the productivity application extender 120 to provide said network-based functionality.

It should be noted that the user tools 330 are intended for the end user. On the other hand, the developer tools 320 are intended for the developer. The productivity application extender 120 may be utilized by both the user tools 330 as well as the developer tools 320. As

such, if for example one created a set of functionality geared at lawyers, called "lawyer tools," the lawyer tools may leverage some of the packaged functions of the productivity application extender 120 in addition to the functionality contained therein. It should be noted that all components of the network-functionality software 210 work in conjunction with the network-enabling objects 130 in the Spreadsheet 110.

According to one embodiment, the user tools 330 are created and packaged as a COM component. The user tools 330 are deployed as one DLL file. According to another embodiment, the user tools 330 may be provided as distinct COM components that can be installed by software developers at their own wish. It should be noted that designing the user tools 330 as a DLL file is just one method of deployment, and various other manners of deployment exist within the scope of the present invention.

The present invention also provides users with the ability to incorporate extended services 340 designed by the third-party software developers. According to one embodiment, the third party developers may create network-enabling objects using the developer tools 320 (FIG. 3A). According to another embodiment, the extended services 340 may be created by any generic tool for developing COM components. The use of extended services 340 provides the flexibility of later augmenting a productivity application 100 with new network-based functionality that may be created or authored in the future, while still being designed in accordance with the information model 790 (FIG 7D). This allows developers the opportunity to leverage the existing functionality available in the productivity application extender 120 or even the user tools 330 or developers tools 320.

The productivity application extender 120, the user tools 330 and the extended services 340 are integrated or coupled to the productivity application 100 through the

productivity application extension enabler 260. This integration may be achieved through standard methods of extensibility. It should be noted that the productivity application 100 may be extended through a variety of means. For example, Microsoft's products can be extended through an "add-in" model, where the COM object extenders, such as the user tools 330, the extended services 340 and the productivity application extender 120, are defined as add-ins at design time, through, for example, the use of a wizard or by implementing a required programming interface.

FIG. 4 depicts one embodiment of the process for providing custom network-based functionality to a productivity application 100 from the network-enabling software 210. The COM component is installed on a PC that has the productivity application 100 running thereon, and may be used in conjunction with the spreadsheet 110 opened in the productivity application 100. The COM component enables the network-enabling objects 130 to receive the network-based functionality in the opened spreadsheet 110.

At step 410, the custom functionality is authored, which may leverage functionality found in productivity application extender 120, the developer tools 320 and the user tools 330. There are various mechanisms to create the custom functionality, such as using Visual Basic® development language to create ActiveX objects, programming a component in the C++ language, and the like. At step 420, the developer of the custom functionality enters certain custom registration information into the object created. The custom information may include, for example, properties such as an object's fixed name, display name, and Global Unique Identifier (GUID). This custom registration information is needed to conform the created custom functionality to work correctly with the rest of the invention's architecture. The custom registration information is used to provide a common way for all of the pieces to be able to

communicate with each other. With correct registration information, a custom set of functionality can link itself to, and make use of, the productivity application extender 120. The productivity application extender 120 then becomes aware of the custom functionality and the services provided, and can manage them appropriately. It is important to conform the created custom functionality appropriately so that the productivity application extender 120 can be aware of and manage the custom functionality. In one embodiment, the custom functionality will make use of the services offered by the productivity application extender 120 as well as use the productivity application extender 120 to communicate with network-enabling objects 130 within the spreadsheet 110.

As shown in step 430, once the authored custom functionality has the appropriate registration information, the functionality is packaged as a DLL file with the desired configuration. Thus, if the developer wishes to deploy a plurality of DLL files in an installation program, the DLL files would be packaged together with an installation manager. It should be noted that the developer can also set other properties on the network-enabling object 130, such as threading models, debugging levels, and the like.

At step 440, the DLL for the appropriately packaged custom functionality is provided to the user. The user may obtain the custom functionality in a variety of different ways, such as in an installation disk or Compact Disk (CD), or via download from a web site. The obtained DLL for the custom functionality is installed on a computing device, as shown in step 450. The installation takes place in accordance with the developer's designed method, such as running an install program to automatically install the DLL on the computing device in the appropriate location.

At step 460, the custom DLL registers with the operating system registry, which allows the DLL to operate/function correctly under the COM architecture. For example, if the DLL is being installed on MICROSOFT, WINDOW, the DLL registers with the WINDOWS registry. The registry provides a secure, unified database that stores configuration data in a hierarchical form, which allows system administrators to easily provide local or remote support, using administrative tools. The registry also allows developers to store their own configuration data for their custom applications.

At step 470, additional information is read from the DLL by a service manager that resides in the productivity application extender 120. This is the information originally entered in step 420 by the developer, dictating the rules for the manner in which the custom functionality is to be utilized.

Finally, in step 480, the service manager of the productivity application extender 120 makes the registered DLL available to the productivity application 100. It should be noted that the DLL has the necessary properties, run-time information and other information needed. At this point, the service manager in the productivity application extender 120 knows that a new set of functionality exists and how to offer the functionality to other system components. Once the productivity application extender 120 is installed, the service manager is available.

Once the necessary productivity application extender 120 and developer tools 320 have been installed, the software developer may use these tools to author custom network-based functionality that can be made available as extended services 340.

It should also be noted that users will utilize the aforementioned process for installing the necessary user tools 330 as well as extended services 340. Once the necessary productivity application extender 120, user tools 330 as well as extended services 340 have been

installed, the users may access network-based functionality in their documents 110 opened within the productivity application 100.

According to the invention, and as discussed above, two logical pieces of code are required for operation of the invention. The first piece is the embedded network-enabling objects 130 and a network-enabling code 630 (FIG. 6). The second piece is the network-enabling software 210, including the productivity application extender 120 and/or user tools 330 and/or developer tools 320. While in one embodiment, the network-enabling software 210 is an external piece of software that augments the productivity application 100, as discussed above, in another embodiment, the network-enabling software 210 can be embedded into the productivity application 100 as an integral part thereof.

FIG. 5 provides an illustration of a productivity application 500 which is configured to provide users with network-based functionality as an integrated feature thereof, in accordance with the present invention. The productivity application 500 may be augmented by the use of one or more generic add-in(s) and/or plug-in(s) 220 that may be supplied by third-party vendors, or the like.

The productivity application 500 comprises a native productivity application code 520. The native productivity application code 520 includes code that makes the productivity application 500 operational and provides the functionality and features that are standard for the productivity application 500. The native productivity application code 520 generally comprises the code that creates and provides the UI 522 for the underlying productivity application 500.

The native productivity application code 520 may also include a productivity application extension enabler 530 which allows and facilitates the use of add-in(s) and/or plug-

in(s) with the productivity application 500. The productivity application extension enabler 530 is a facility that allows generic add-in(s) to plug into the productivity application 500 for adding new functions, operations or features that were not originally available in the productivity application 500.

The native productivity application code 520 may expose a native object model 540. The native object model 540 includes a description of an object architecture, including the details of the object structure, interfaces between objects and other object-oriented features and functions. The native object model 540 enables the use of an add-in 220 or any other macro or extender running on the PC so that the functionality of the add-in 220 and/or other macros or extenders can seamlessly be integrated into the productivity application 500. The native productivity application code 520 may also include an API 544 to allow the productivity application 500 to interact with other programs for expanding the standard features or providing additional functionality thereto. According to one embodiment, the API 544 is an integral part of the native object model 540.

The native productivity application code 520 may provide a native script engine 550. The native script engine 550 is a facility in the productivity application 500 that executes macros or any other extenders that are to be run along with the productivity application 500.

The native productivity application code 520 is provided with code that provides network functionality. The code for network functionality 560 comprises a plurality of network-based utilities. The code for network functionality 560 may be configured to handle application services, such as XML parsing and handling, session management, security, event handling and service management, location management, query routing, document packaging, caching, data initialization and persistence, command routing and other similar functions and operations that

assist in network-based activities. In addition, the network-based utilities are configured to provide various optional utilities that are used to create Spreadsheets 110 with network-based functionality therein. The network-based utilities may include layout utilities, forms management, query management, data modeler, packaging assistant, page scraping, stock quotes in real time, and other miscellaneous utilities and operations that assist in network-based activities within the spirit of the present invention.

Using the available features of the productivity application 500, one can create a new spreadsheet 110 or open an existing spreadsheet 110 to perform operations thereon. The code for network functionality 560 allows users to embed network-enabling objects into the spreadsheet 110, which will be described in greater detail below. Once the network-enabling objects are in the spreadsheet 110, the code for network functionality 560 allows a user to use the network-enabling objects and receive the network-based functionality in their spreadsheet 110, in addition to the non-network functionality and features provided by the productivity application 500.

FIG. 6 provides an illustration of one embodiment of a spreadsheet 110 in accordance with the present invention. The spreadsheet 110 comprises document code 610 and content 615. The document code 610 includes all computer readable instructions necessary to provide the Spreadsheet 110 with its look and feel, as well as provide support for any functionality embedded in the spreadsheet 110.

The document code 610 includes a script code 620. The script code allows users or developers to extend the built-in functionality of the productivity application 500 or simply create macros containing often-used packaged sequences of operations (for example, repeatedly creating the same customized chart from a set of data). The script code 620 also may provide the

ability to interact with ActiveX controls and other COM components on the computing device to utilize their provided functionality within the spreadsheet 110.

The document code 610 may further include network-enabling code 630 for providing network-based functionality within the spreadsheet 110 at launch-time by initializing and launching the network-enabling object 130 to provide network-based functionality. According to one embodiment, the network-enabling code 630 is unpacked and read by the productivity application extender 120 at launch-time, and after that the network-enabling code 130 is not used, accessed, and/or read during run-time. In other words, during run-time of the network-based functionality, the network-enabling code 630 plays no role, and is akin to not being present. The network-enabling code 630 is unpacked (i.e., read) at load/launch time by the productivity application extender 120 to support the functionality of the productivity application extender 120.

The network-enabling code 630 contains computer readable instructions for allowing the initialization and/or launch of the network-based functionality as well as any other functions within the scope of the present invention. According to one embodiment, the network-enabling code 630 may include a plurality of routines to provide different operations and functionality. According to another embodiment, the network-enabling code 630 may be a plurality of distinct modules, separately packaged in the document code 610. The network-enabling code 630 may be programmed to include support for query instructions, layout instructions, data modeling, and various other features that may augment the spreadsheet 110 with additional network-based functionality.

According to one embodiment, the network-enabling code 630 is packed together in a secure location within the document code 610 of the spreadsheet 110. In the event that the

network-enabling code 630 is packed, it must be unpacked prior to use of network-enabling code 630 for providing network-based functionality. Furthermore, the network-enabling code 630 may be encrypted with a suitable encryption algorithm to ensure that the network-enabling code 630 is tamper-proof, as well as to restrict utilization of the underlying network-based functionality to the appropriately registered modules. According to one embodiment, the network-enabling code 630 is packed for storage and/or transmission over a network or the Internet.

It should be noted that when the spreadsheet 110 with network-based functionality is stored and/or uploaded, the spreadsheet 110 can have a mix of packed and unpacked network-enabling code 630. The packed portion of the network-enabling code 630 requires unpacking prior to run-time of the network-based functionality.

The content 615 in the spreadsheet 110 includes data 660 and network-enabling object(s) 130. The data 660 may include static data 670 as well as dynamic data 680. Static data 670, once entered, maintains its state and is often only altered by a user. Dynamic data 680, on the other hand, may be added and/or altered at any time, possibly without any active intervention by a user. For example, the spreadsheet 110 could be configured for allowing the dynamic data 680 to be altered in real-time by content from the Internet or other networks by providing a real-time stock feed into the spreadsheet 110.

It should be noted that according to one embodiment, the content 615 may also be provided as a mix of packed and unpacked content in addition to the network-enabling code 630. For example, where it is desired that the user not be able to access certain pieces of content, the content may in fact not get completely unpacked at runtime. Similarly, it is possible to not unpack part of the network-enabling code 630 to deprive an unlicensed user of its functionality,

and the still-packed code 630 would be inert and not executable at runtime. Additionally, it is possible to pack the content 615 in an encrypted format.

The network-enabling object 130 may be embedded as part of the content 615 of the spreadsheet 110. As part of the invention's distributed architecture, the network-enabling object 130 may be able to communicate with the script code 620. Through the script code 620, the network-enabling object 130 may communicate with the productivity application extender 120, and the productivity application extender 120 then communicates with the network-enabling code 630. It should be noted, that the network-enabling object 130 is also capable of interacting directly with the productivity application extender 120.

According to one embodiment, the network-enabling object 130 may be a self-contained packaged set of functionality. According to another embodiment, the network-enabling object 130 will operate in conjunction with the productivity application extender 120 and other custom functionality that persists on the client, such as user tools 330, developer tools 320 and/or other add-ins.

According to the invention, the developer may leverage the distributed architecture and decide how "heavy" to make the network-enabling object 130 portion of the code, by providing all the requisite code for receiving network-based functionality as part of the network-enabling object 130. The developer can decide the amount of functionality to be put into the network-enabling object 130, versus putting the network-based functionality as a custom add-in. The best balance will depend on the particular functionality being deployed, and perhaps even on deployment issues such as code size for add-in code being downloaded from the Internet for installing on the client.

FIG. 7A-7D provides an illustration of different states of communication between the spreadsheet 110 with one or more controls external thereto. According to an embodiment of the present invention, the spreadsheet 110 comprises a wireframe 705, which has content 615, native script code 620 and network-enabling code 630. The content 615 comprises user interface (UI) 710, which has information regarding formatting and layout for the data 660 and the objects 130 therein. The UI 710 essentially provides the look and the feel of the content 615. For example, the UI 710 controls the fonts, formatting and the layout of text in the spreadsheet 110. The content 615 further includes configuration 720, which enables and facilitates the user's viewing thereof. For example, the configuration 720 may allow a user to decide the zoom-in level of the spreadsheet in a MICROSOFT EXCEL spreadsheet.

As noted above, the content 615 includes static and/or dynamic data, depending on the choice of the user. In addition, the content 615 is provided with network-enabling objects 130 for launching network-based functionality from within the spreadsheets 110.

According to the invention, the native script code 620 provides the ability to interact with ACTIVE X controls and other COM components on the computing device to utilize the network-based functionality within the spreadsheet 110. The native script code 620 permits the productivity application 100 to leverage its core functionality and permits the productivity application 100 to provide network-based functionality from within the productivity application 100 environment. While there are various network-based functionality that can be provided, they may include communications services that allow a user to communicate and interact with others, streaming services that allow a user to receive and play streaming content, data services that allow a user to retrieve, manipulate, and view data retrieved over a network, as well as various other services within the scope of the present invention.

According to one embodiment of the present invention, the native script code 620 allows any of the network-enabling objects 130 in the spreadsheet 110 to interact with the productivity application extender 120 running alongside the productivity application 100. More particularly, external components communicate with the network-enabling objects 130 via the productivity application extender 120. While this is one mechanism of operation, it should be noted that the present invention also allows the productivity application extender 120 to directly communicate with the network-enabling objects 130, without requiring any intervention from the native script code 620, according to another embodiment of the present invention.

The network-enabling code 630 facilitates the providing of network-based functionality within a spreadsheet 110. The network-enabling code 630 contains computer readable instructions for launching network-based functionality as well as any other functions within the scope of the present invention. The network-enabling code 630 may be programmed to include support for query instructions, layout instructions, data modeling, and various other features that may augment the spreadsheet 110 with additional network-based functionality. As noted above, the network-enabling code 630 may be packed in a secure location within the spreadsheet 110, in which case it must be unpacked prior to its use. Furthermore, the network-enabling code 630 may be encrypted.

According to the invention, the native script code 620 is capable of communicating with the productivity application extender 120 or other external COM components 780 or ACTIVE X controls 770. According to the invention, there are different models according to which the network-enabling objects 130 may interact with the productivity application extender 120 to provide network-based functionality, as shown in FIGS. 7A-7D.

Accordingly, a developer may extend and customize the productivity application extender 120 in a variety of ways to provide any additional type of customized functionality.

When designing additional ActiveX 770 and/or COM controls 780, the developer can have them work in conjunction with network-enabling objects 130, allowing the productivity application 100 to become a distributed application. The developer has the flexibility to design and provide network-based functionality in different ways, since they can utilize the productivity application extender 120, or they can provide the network-based functionality via the native script code 620. According to one embodiment, a part of the distributed productivity application manifests itself as the network-enabling object 130 and another part takes the form of an ACTIVE X control 770. According to another embodiment, another part of the distributed productivity application could be a COM control 780 that leverages functionality available in the productivity application extender 120.

FIG. 7A provides an illustration of the present invention at launch-time of the network-based functionality. Initially, network-based functionality is launched when the productivity application extender 120 reaches into the Spreadsheet 110 and unpacks (i.e., reads) the network-enabling code 630. The information from the network-enabling code 630 is stored in the productivity application extender 120 which allows it to provide the appropriate network-based functionality at runtime. It should be noted that the productivity application extender 120 is not supported by the other external COM components 780 or ACTIVE X controls 770, initially. Furthermore, in this embodiment, the productivity application extender 120 does not need to communicate with the native script code 620 to unpack the network-enabling code 630.

As shown in FIG. 7B, according to one embodiment, the network-enabling object 130 communicates with the PRODUCTIVITY APPLICATION extender 120 to provide

network-based functionality in the Spreadsheet 110. According to this embodiment, the network-enabling object 130 is provided with the necessary code to interact with the productivity application extender 120 to launch, obtain and utilize the network-based functionality within the Spreadsheet 110. It is possible for the productivity application extender 120 to interact with the COM component(s) 780 to seek additional functionality not available in the productivity application extender 120, as indicated by the instructions contained in the network objects code. It should be noted that the network-enabling code 630 play no role in the process. According to this embodiment, the native script code 620 is not necessary for providing the network-based functionality.

As shown in FIG. 7C, according to another embodiment, the productivity application extender 120 communicates with the network-enabling object 130 to provide network-based functionality in the Spreadsheet 110. Furthermore, the productivity application extender 120 may interact with the COM component(s) 780 to seek additional functionality not available in the productivity application extender 120. Based on the operations and features of the productivity application extender 120 and/or the COM components 780, the native script code 620 may be provided with the requisite information to allow it to communicate with the network-enabling objects 130. According to one embodiment, the native script code 620 can communicate directly with the ACTIVE X 770 and/or COM controls 780 to provide additional functionality not available through the productivity application extender 120. As a result, the network-enabling objects 130 may provide network-based functionality in accordance with the PRODUCTIVITY APPLICATION extender 120 as well as the ActiveX 770 and/or COM controls 780.

As shown in FIG. 7D, according to another embodiment, the network-enabling object 130 communicates with the productivity application extender 120 to provide network-based functionality in the Spreadsheet 110. Furthermore, the information from the unpacked network-enabling code 630 that is read by the productivity application extender 120 provides the requisite information to the native script code 620, and creates and provides the necessary information to an information model 790. Where the network-enabling code 630 is provided packed, the productivity application extender 120 unpacks the network-enabling code 630 prior to launch to read the necessary information therefrom and create the information model 790. The information model 790 is a holding area that houses information about the functions the spreadsheet 110 is supposed to support at runtime, and the methodology for running the functions. For example, the information model 790 might contain a query definition of how to retrieve stock quote information, namely the server to look for, the data parameters to retrieve, and the mechanism for handling the resulting data sets.

The information model 790 interacts with the network-enabling objects 130 to provide the appropriate network-based functionality within the spreadsheet 110.

Since network-based functionality may be distributed between network object 130 and COM component 780, the desired network-based functionality can be authored by a developer according to the best suited structure. A developer may choose to place more functionality in COM component 780, resulting in a "lightweight" network-enabling object 130. This allows easier distribution of the spreadsheet 110 once the base COM component 780 has been installed.

FIG. 8 is a schematic flow diagram depicting the process for creating a spreadsheet document with network-enabling objects embedded therein. In general, the objects

130 may be embedded into spreadsheets 110 in several ways. They can be placed into documents by developers, and the documents can then be posted on a web site or e-mailed to users. When coupled with the ability to control and manage the behavior of the containing application, this basically provides for the development of powerful, distributed "embedded applications" that are essentially defined by the combination of the productivity application and the enhanced spreadsheet within it. Alternatively, the network-enabling objects 130 can be made available to users and stored on user systems. Users can then place objects personally into individual spreadsheets 110 they create or receive. This capability can be used in combination with other Internet or network functions.

The present invention may also be used to provide professional developers with a tool set for creating productivity application based services and applications, such as "Productivity application Web Sites." While certain development capabilities may best be provided through a specific, separate application program for developers using the technology, many capabilities may be provided to developers through an extended version of the target productivity application itself. Thus, for example, this will allow a high fraction of web site development to be conducted using the familiar, functionally rich, and domain-specific environment of a productivity application rather than traditional web site development tools or lower level development environments such as C or Java. This will facilitate the workflow processes associated with web site development, by permitting partially completed productivity application -based web sites to be e-mailed, viewed, and edited by multiple developers and users, using augmented versions of a single productivity application. Moreover, where business rules can be defined using the built-in script and/or native functionality of the document and desktop application, parts of the productivity application-based web site can essentially be coded directly

by domain experts (i.e., users) rather than passing specifications off to a developer for implementation, thereby potentially reducing development time and cost.

Referring to FIG. 8, at step 810, the productivity application 100 is launched, along with the network-enabling software 210. According to one embodiment, because of the appropriate configuration of the network-enabling software 210, it automatically launches when the productivity application 100 is launched. At step 820, the Spreadsheet 110 is opened. According to the invention, the Spreadsheet 110 may be new and completely blank, initialized through a pre-defined template, or a pre-existing document.

Once the spreadsheet 110 is open, data may be created therein, as shown in step 830. For example, content for a word processing document may be entered by the user. As noted above, the data may be static or dynamic. In step 840, network-enabling objects 130 are embedded as part of the content 615 of the spreadsheet 110. There are a variety of ways in which users may be allowed to embed the network-enabling objects 130 in the spreadsheet 110. According to one embodiment, the user is provided a toolbar as the UI of the network-enabling software 210, from which the user can simply choose the desired network-enabling object 130, and then drag and drop in the appropriate location of the spreadsheet 110.

As one embodiment, the system allows a user to encrypt the network-enabling code 630, in step 850. In step 860, the network-enabling code 630, which may or may not be encrypted, is placed in the document code 610 area of the spreadsheet 110. According to one embodiment, the network-enabling code 630 may be packed in a hidden area so that it is invisible to the user. According to another embodiment, the network-enabling code 630 may be compressed before being placed in the document code 610 area of the spreadsheet 110.

Finally, in step 870, the spreadsheet 110 is saved on the local system. As the above discussion shows, the saved spreadsheet 110 comprises data as well as network-enabling objects 130 and network-enabling code 630. During runtime, the network-enabling objects 130 and the native productivity application code 230 interact with the network-functionality software 210 to provide network-based functionality in the spreadsheet 110.

FIG. 9 provides one embodiment of the process for loading the spreadsheet 110 with network-based functionality therein. At step 910, the productivity application 100 is launched by a user. As depicted in box 920, once the productivity application 100 has been launched, the user opens a standard spreadsheet for the launched productivity application 100. For example, if the launched productivity application is MICROSOFT EXCEL, then the open document will be an EXCEL spreadsheet or any other compatible EXCEL file type.

At step 930, the installed productivity application extender 120 detects the opening of the spreadsheet 110. At step 940, the installed productivity application extender 120 inspects to determine if the spreadsheet 110 has the appropriate network-enabling objects 130 and the network-enabling code 630 therein for providing network-based functionality within. At step 950, based on the inspection in step 940, the system determines whether the spreadsheet 110 has the appropriate network-based functionality therein. If the system determines that the launched Spreadsheet 110 does not have the requisite network-based functionality, then the spreadsheet is opened with no intervention from the productivity application extender 120, as shown in step 960.

On the other hand, if the system determines that the launched spreadsheet 110 has the requisite network-based functionality therein, then the network-enabling code 630 is located by the productivity application extender 120 in step 965. If the network-enabling code 630 is

encrypted, then the productivity application extender 120 causes decryption of the network-enabling code 630 to read it, as shown in step 970. At step 980, the system initializes and launches the spreadsheet 110 along with the network-based functionality in accordance with the activation of the network-enabling objects 130 and/or the instructions provided in the network-enabling code 630.

FIG. 10 provides an exemplary illustration of a client/server architecture utilizing the network-enabling features provided to the client, in accordance with the present invention. A client 1000 hosts the underlying productivity application 100. The spreadsheet 110 is launched within the productivity application 100. As noted above, the spreadsheet 110 is provided with one or more network-enabling objects 130, which communicate with the productivity application extender 120 running with the productivity application 100.

The client connects to a network 1050 for receiving the network-based functionality for which the network-enabling objects 130 are configured. Using the network 1050, the client 1000 may connect with one or more network servers, such as a web server extender 1060 which runs on a generic web server 1065, or a specialized server(s) 1070 and/or one or more generic third-party servers 1080 for real-time feeds. In one implementation, the client 1000 may be provided with one or more communication means for connecting and interacting with the network 1050. The communication means is coupled to a central processor on the client machine. The scope of the present invention is not limited by the choice of communication means employed. Accordingly, it is possible to use one or more modems, a Digital Subscribe Line (DSL), an integrated Service Digital Network (ISDN) line, a cable modem or any similar device to connect to the server and communicate therewith.

The clients are able to connect to the server and receive their network-based functionality. For example, a client 1000 may connect to a web server extender 1060 which connects to a generic web server 1060 for receiving regular feeds about fluctuations in the stock market, or connect to the specialized server(s) 1070 that provide specialized services to the client, such as real-time feeds 1085, collaboration support 1090, data access 1095, workflow servers, licensing servers and the like. If a client connects to the generic third-party servers 1080 for real-time feeds, the client system is provided with specialized network-based functionality in the spreadsheet 110 that allows the extended productivity application 100 to utilize the real-time feeds within the spreadsheets 110.

It should be noted that the client/server architecture of FIG. 10 can be utilized to download individually or collectively the productivity application 100, the Spreadsheet 110, network-enabling software 210, and/or the network-enabling object 130 from the Internet or some other compatible network. For example, a request for a download may be received at the web server extender 1060 which appropriately transmits it to the web server 1065, or the request may be received at the specialized server(s) 1070. Accordingly, the client may transmit a login request, and then make a request to download the desired component(s) of the present invention. The request is appropriately handled at the receiving server, and the request is processed by the server. Finally, the server serves the requested component(s) of the present invention to the remote client. According to one embodiment, the client may be required to authenticate its identity prior to the server serving the requested component.

The web server extender 1060 may be utilized for custom request handling of client requests. According to one embodiment, the custom request handling may be

authenticating the remote client. According to another embodiment, the custom request handling may be deciding the version of the requested component(s) to be served to the remote client.

For example, web server extender 1060 may provide intelligence to determine appropriate version of a financial data sheet to return to a client – either a productivity application spreadsheet version or an HTML web page version.

FIGS. 11A-11B provide an illustration of the present invention where spreadsheets are provided with network-based functionality therein. It should be noted that the spreadsheets described in FIGS. 11A-11B are merely two embodiments of the present invention, and various other embodiments may be constructed using the disclosed invention.

As shown in FIG. 11A, the spreadsheet 1100 relates to a financial spreadsheet with complete information of a user's finances and comprises various components as part of its content. At the top of spreadsheet 1100, there is a page title 1105, which is a part of the static data. The content further comprises a summary chart 1110, which contains information regarding a plurality of financial accounts held in the user's name. Each financial account is listed individually, and may be provided as a link to another document, web site, page within the same document, and/or application to provide users with more detailed information for each entry.

The content may also include formulas that compute mathematical values, such as the one shown as the total current value 1120 such formulas being normally utilized by preserve productivity applications. According to the present invention, various charts providing graphical depictions 1125 of the data shown in the summary chart 1110 may be provided as part of the content of the spreadsheet 1100.

According to the example depicted in FIG. 11A, each of the financial accounts are provided as a link to another account accessible over the network. Thus, clicking on E*TRADE takes users to the appropriate page that maintains information about the user's stock portfolio; clicking on CITIBANK Checking takes users to the appropriate page that maintains information about the user's bank account; clicking on CITIBANK VISA takes users to the appropriate page that maintains information about the user's credit card, and similarly other links transport the user to the appropriate page.

FIG. 11B provides an illustration of a spreadsheet 1130 that contains a plurality of network-based functionality therein for providing information regarding the user's stock portfolio. As shown in FIG. 11B, the spreadsheet 1130 relates to a financial spreadsheet with complete information of a user's stock portfolio. The spreadsheet 1130 comprises a wireframe, population data, and form data. The wireframe includes static content and native script code that is generally not modified during runtime within a productivity application residing on an end-user's system.

Population data is dynamically generated data used to populate specified areas within a wireframe during the productivity application's runtime. Generally, population data is supplied by a server and is designated "read only" and is not altered. Population data may be used to populate fields, strings, and pick lists for display to the user, or transparently as calculation data operated on by the application. Population data may also be defined to be a function of information maintained in the associated wireframe, user input, or both, and may typically be generated or retrieved upon actions such as startup, connection, form submission, or in real-time in response to remote data changes.

Form data is data that is specified or manipulated at the client computer by users during runtime for the eventual purpose of transmission to a server. Form data is usually one-way data transmitted from client to server. But in some circumstances it may be desirable to read such data from a server (for example, as cached prior information to pre-populate a user form or as smart defaults for a user form) to pre-fill forms within a wireframe. Form data includes data items, such as data entered into text box prompts (for example, asking for a user's username and password for authentication) and parameterized data entered into defined queries (such as stock ticker symbol as a parameter in a we b query).

The spreadsheet 1130 comprises a market chart 1135 that is connected to a network so that it receives historical data for the market index in real time. The market chart 1135 forms a part of the population data that is read-only and cannot be changed by the user. A toggle button 1140 is provided to switch the market chart 1135 between display of historical data for each of the selected indices.

The spreadsheet 1130 further includes a portfolio table 1132 for the user, which contains detailed information regarding the various stocks owned by the user. The portfolio table 1132 has layout definition as part of the wireframe, and includes static data such as the heading "Portfolio", heading for various columns, such as "Symbol," "Current Price ($)", and the like, as well as native script code that is not visible to the user. The native script code includes instructions for fetching the information regarding the stock's trading volume, stock's price, number of shares owned and the like, based on the identity of the user. Information regarding shares owned by the user represents form data and is transmitted from the client computer to the server. On the other hand, displayed information regarding stock's price is population data and is supplied by the server to the client user; this information is read-only.

Network-enabling objects in the form of Alert Wizard button 1142 and Trade button 1144 are provided in the wireframe. The user may click the Alert Wizard button 1142 to launch a network-based functionality that allows the user to receive appropriate alerts and notifications on a regular basis. The user may click the Trade button 1144 to launch a network-based functionality that allows the user to make stock trades over the network. Behind-the-scenes functionality, e.g. trades, is provided either by the productivity application extender 120, network-enabling object 130, or by a custom COM object.

Furthermore, the spreadsheet 1130 includes a web browser 1150 that is embedded in the spreadsheet 1130 in accordance with the present invention. According to the invention, the web browser 1150 is a network-enabling object that brings network-based functionality within the spreadsheet 1130.

FIG. 12-14 illustrate one exemplary embodiment for a system and method for both developing and accessing a web page as an embedded application that can be produced using an augmented spreadsheet program and an enhanced spreadsheet document. An example of such an enhanced spreadsheet in provided in FIG. 11B.

FIG. 12 is an illustration of a client/server architecture in accordance with one exemplary embodiment of the present invention, which provides a system and method for both developing and accessing a web page as an embedded application that is produced using the spreadsheet productivity application 1200 that has been extended with a network-functionality software 1220 and a network-enabled spreadsheet document 1210. The embedded application web page allows an end-user to enter, for example, a stock's symbol, receive relevant historical data relating to the stock via a web server, and chart the received data within the embedded

application web page using the built-in charting facilities of the host spreadsheet application 1200.

The developer's system comprises a developer client computer 1202 that is connected to network 1204 via a communication link 1206. Network 1204 may, for example, be a private intranet, a virtual private network, or the global Internet, and the communication protocol used on the network may be TCP/IP or any protocol layered on top of IP, such as SMTP, FTP, HTTP, etc. In this example, the developer client computer 1202 is provided with a spreadsheet productivity application 1200, such as MICROSOFT EXCEL or LOTUS 123. The client computer 1302 is also preferably provided with a developer's network-functionality software 1220 that adds tools and/or provides functionality to the spreadsheet productivity application 1200 to facilitate the creation of spreadsheet web pages, and an end-user's network-functionality software 1250 for providing runtime support for the network-based functionality within the spreadsheet documents.

The developer is connected to a web server 1214 via the network 1204. The web server 1214 is connected to the network 1204 via a communication link 1216. Web server 1214 provides access via said network 1204 to one or more web pages 1234 or customized embedded application web pages 1236. The original spreadsheet document is now a custom embedded application 1236. Web pages 1234 or customized embedded application web pages 1236 can be stored locally or any place on any network that the web server 1214 can access.

The end-user's system comprises a client computer 1240 that is connected to the network 1204 via a communication link 1206. The client computer 1240 is provided with end-user's network-functionality software 1250, which may include user tools and a productivity application extender.

It should be noted that the developer is provided with tools that may be distinct from those provided to a end-user, because the developer uses the system to create an embedded application web page 1236, while the end-user uses the system to access and interact with that embedded application web page 1236.

In summary, the present system and method relates to a developer at client computer 1202, who identifies an HTML page 1234 to be rendered as an embedded application web page. The developer imports a snapshot of the HTML page 1234 into spreadsheet software 1200 and onto the first sheet of a new spreadsheet 1210. The developer may then modify the web page's layout to improve its appearance so that it is more appropriate to a spreadsheet-based application.

The HTML page being rendered will typically include some content that changes infrequently, if at all, and other data or content that changes in response to user actions, remote data updates, and so forth. For example, a web page maintained by an information provider that displays stock prices information will generally include some information that remains essentially constant, such as the page layout, ticker symbol, information provider's name and logo, as well as other information that requires frequent updating, such as stock prices and performance data. The present invention distinguishes between these "static" and "dynamic" content elements, and manages them in different ways.

The static portions of the HTML Web page are preferably rendered in the spreadsheet version of the page as static data within one or more cells or controls within the UI of the embedded application web page. In one embodiment, static HTML text and images may be incorporated in the spreadsheet document using the base HTML import operation of spreadsheet productivity application 1200, if included.

According to one embodiment, the dynamic portions of the HTML Web page are rendered as one or more named cells, charts, or other controls. The developer links these UI elements to state-information elements that may, for example, be stored in cells on a second spreadsheet page or elsewhere in computer memory. These state-information elements can themselves be linked to one or more queries stored, for example, in cells on a third spreadsheet page or elsewhere in memory or the information model 790. At runtime, these queries are executed and the query results are stored as state information in the linked state-information elements. This state-information is then used to populate the dynamic portions of the web page within the spreadsheet.

It may also be desirable to include form elements to allow an end-user to enter "form data," such as the stock symbol whose price the end-user wishes to retrieve. These form elements are preferably rendered as one or more cells or other controls, such as entry fields, pick lists, and the like, on the first spreadsheet page 1210. The developer links each such UI form element to one or more state-information elements preferably stored in cells on the second page of the spreadsheet 1220 or elsewhere in memory. Information entered by a user in a form element on the first spreadsheet page is passed back to these linked state-information elements for further processing or transmission to web server 1214.

The developer then refines the embedded application UI by hiding distracting elements that will not be needed by users in typical operation of the page 1210, such as secondary scratch sheets and extraneous toolbars. Using tools provided by the invention, the developer may decide to restrict or otherwise control the extent to which an end-user may modify or interact with the new web page. The embedded application web page 1210 is then placed into run mode which may prevent any disruption of the UI by users. The completed

enhanced spreadsheet web page 1210 is then posted to web server 1214 as custom embedded web page 1236. This deployment may include providing links to the embedded application web page 1236 from other pages, such as the HTML page 1234, on the same or other web sites.

A user at computing device 1240 may download the embedded application web page 1236 from web server 1214 by opening the page in spreadsheet software 1260 directly or by its hyperlink, which also causes the page to automatically be opened in spreadsheet software 1260 as the enhanced spreadsheet document 1270. The network-functionality software 1250 and the runtime code in the enhanced spreadsheet document 1270 cooperate to execute any queries to retrieve population data elements as specified by the developer. The contents of these elements are used to populate the dynamic data regions of the first spreadsheet page 1270, which may then be displayed to the user.

While viewing the enhanced spreadsheet document 1270, the user has available all functionality provided by spreadsheet software 1260 that the web page developer has left accessible. For example, if the displayed page contains raw stock data, the end-user may use the spreadsheet software's chart functionality to create a chart from said data. In addition, a scratchpad area may be provided on an additional sheet of the spreadsheet that allows the user to perform arbitrary calculations and manipulations based on the data displayed on the web page. The user may also link to data on the web page from a second spreadsheet or other program running on computing device 1240. Furthermore, if the user enters a new stock symbol to be charted, the relevant data will be returned to the client via a population data query and charted, causing only the updated areas of the client-side embedded web page 1270 to be updated. As a result, the present system provides high update and refresh performance. It should be noted that custom analysis can be performed locally as opposed to constant calls to the network.

FIG. 13 provides a provides a detailed flow of the operation by a developer to create a web page in accordance with the present invention, using a spreadsheet productivity application 1200 that has been provided with a network-functionality software 1220. In step 1302, the developer launches spreadsheet productivity application 1200. Launching the spreadsheet productivity application 1200 also launches the network-functionality software 1220, including the developer tools and user tools. In this embodiment, developer tools of the network-functionality software 1220 add a developer's toolbar to spreadsheet productivity application 1200 for inserting macros and UI elements into the spreadsheet document 1210. These tools may be made available on one or more developer's toolbars or menus that become embedded in the software's UI.

In one embodiment, the toolbar may include a "new Web page" button for creating a new spreadsheet web page by providing a rendering of a web page as an enhanced spreadsheet document. In step 1304, the developer clicks on the "new Web page" button. In step 1306, the system presents to the developer a wizard containing a palette that displays a selection of templates for designing a web page. In addition, the wizard includes a "browse to web page" button.

According to the present invention, it is assumed that the developer wishes to create a spreadsheet web page that "mirrors" an existing browser-based page, rather than create a spreadsheet web page from scratch. Accordingly, in step 1308, the developer clicks on the "browse to web page" button on the developer toolbar. In step 1310, a browsing window pops up within the spreadsheet's UI and the developer browses to and selects the web page that the developer wishes to copy. According to one embodiment, the developer browses to a web page 1234 displaying stock price performance.

In step 1312, a snapshot of the selected web page is imported onto the first sheet of the spreadsheet document 1210. This step may employ the native HTML import functionality of the spreadsheet productivity application 1200.

As those skilled in the art will recognize, the native HTML import functionality of spreadsheet programs typically create layout errors, including overlapping and obscured items. Consequently, in step 1314, the developer may be provided with a tool for cleaning up any layout problems created by the format translation. Alternatively, the developer tools in the network-functionality software 1220 may be provided with its own HTML import functionality that does not introduce these layout problems. However, it should be noted that the developer laying out a spreadsheet web page should preferably visualize the page in terms of the optimal layout for the spreadsheet application environment, rather than attempting to copy exactly the look and feel of the page's browser-based version. However, this should preferably also be balanced with the desire to maintain as consistent a look and feel between the two versions as possible.

In step 1316, the developer defines state information associated with the embedded application web page 1210. Each state-information element is implemented as a named property, using developer tools provided to create and manage the properties. These elements may be stored as data values in spreadsheet cells on a second spreadsheet page that is accessible to the developer but may not be accessible to the user. Alternatively, state information may be stored elsewhere in memory or the information model 790 and referenced only by name. Using the state-information manager, the developer defines individual properties of the state information element and optionally specifies a data type, such as string, numeric, dollar, date, and the like, and a default value for the property. Once defined, the document code

in the spreadsheet document 1210 and/or network-functionality software 1250 allows the element to be referenced elsewhere in the application by property name.

In step 1318, the developer uses the developer tool that is provided to define one or more queries for retrieving dynamic data at runtime. These query specifications may be static, i.e., contain no variables, or may alternatively be a function of one or more named state-information elements. These queries and query results may also be located in cells on a third spreadsheet page or alternatively stored elsewhere in memory.

A query manager may be provided as a developer tool, for facilitating the creation and management of these queries. When the developer wishes to define a query, the query manager creates a new data item and displays a dialog box that prompts the developer to enter the desired query. For example, if the developer wishes to create a query that retrieves the stock price information, the developer might enter a URL query and specify that data returned by the query should be stored in a data item that may be labeled *StockData*.

In an exemplary, the "%stocksymbol%" sub-string may be used to indicates that the current value of the *StockSymbol* state-information element should be substituted into the query before it is executed at runtime. This symbol substitution and name reference capability is provided to the end-user by user tools in the network-functionality software 1250. A "data=raw" sub-string may be provided to indicate to the web server that it should return the raw data, possibly encoded in an HTML table. At runtime, the query will be executed by end-user network-functionality software 1250 and the returned data stored in the *StockData* data item, which may then be referenced by name in other fields and controls in spreadsheet document 1210.

In addition, the query manager may be adapted to automatically create a query to retrieve data identified by the user. According to one embodiment, the query manager may permit a developer to direct a browser to a web page and select, for example, a field on the web page with a mouse. The query manager may then automatically formulate an appropriate query for retrieving data through end-user tools in the network-functionality software 1250.

In step 1320, the developer links dynamic-data and form-data values and other fields on the first spreadsheet page to particular state-information or user interface elements. For example, the developer may link the *Stock Symbol* property defined above to the entry field on the spreadsheet web page. It should be noted that the system may be adapted to allow these elements to be referenced by name in formulas and macros so that the developer need not be aware of where or how they are stored.

In step 1322, the developer creates the spreadsheet application-based chart that will replace the static image used in the original HTML page, which is generated during the import step 1312. In particular, the developer deletes that static image from the first page of the spreadsheet document 1210, selects the area of the page that the chart should occupy, and launches the chart wizard of the spreadsheet application 1208. In the wizard, the developer selects one of the standard line or area charts and then specifies the data source for populating the chart, which in this case is the *StockData* data item described above. This connects the chart to the dynamic data stored in the *StockData* data item. The developer then sets the chart's display properties, such as scaling, labels, axis details and so forth, as desired.

In step 1324, the developer may lock or otherwise disable portions of the UI that will be presented to users when spreadsheet document 1210 is opened during runtime. During this step, the developer selects areas of the first spreadsheet page that should be read-only and

presses a "freeze" button on the developer's toolbar. This locks the selected portions of the page so that they cannot be selected and manipulated by the user. Also during this step, the developer may specify spreadsheet pages as invisible to the user. In the present embodiment, the developer will typically designate only the first spreadsheet page as visible. In addition, the developer may wish to hide any standard spreadsheet toolbars that are inappropriate for the particular spreadsheet web page.

In step 1326, the developer presses a mode toggle button provided on the developer toolbar, causing the end-user tools to take over the spreadsheet application and enforce the embedded application web page behavior that has been designated by the developer. The developer may then work locally on the client machine with the web page in runtime mode to verify that the embedded application web page functions properly.

In step 1328, the developer posts the spreadsheet application web page to the web server 1214 and provides a link to the page to allow users to access it. For a more seamless and integrated experience, the developer may additionally utilize a redirector module that runs on the web server in response to end-user requests for the stock web page and transmits either the browser version or spreadsheet version of the web page to a particular user based on any of a number of possible criteria.

FIG. 14 provides a detailed flow of the operation by a user to use a web page created in accordance with the present invention. In step 1402, the user clicks on or otherwise opens a hyperlink associated with a spreadsheet web page 1236. This hyperlink may be a direct link to the enhanced web page 1236 or a link that invokes redirector software at Web server 1214. In the latter case, the redirector software can return either the browser web page 1234 or the enhanced spreadsheet page 1236 based on either client capabilities – namely, whether or not

spreadsheet productivity application 1200 and user network-functionality software 1250 are available on client machine 1240, or on preferences for the specific user. Such preferences may be based on the type of enhanced document or on the particular page or on the web site as a whole or upon other controlling criteria. The page is transmitted to user's client machine 1240 in step 1404 and automatically opened within spreadsheet software 1260. Where enabled, this operation may open the spreadsheet application within the launching browser container using available systems mechanisms such as Active Document Servers and multiple-document interfaces (MDI).

In step 1406, user network-functionality software 1250 and runtime code in the downloaded embedded application web page 1270 cooperate to execute any queries in the web page and to store the returned data in one or more population data elements. In step 1408, this state information is used to populate any designated dynamic-data regions of the first spreadsheet page. In step 1410, the complete "web page" is displayed to the user in the productivity application format. As noted above, while viewing the page, the end-user has access to all built-in spreadsheet functionality of spreadsheet productivity application 1260 allowed by the developer. As a result, the user can still do cell analysis, refer to cells, create charts and figures, and the like.

In some implementations of this embodiment, end-user network-functionality software 1250 and runtime code in web page 1236 may cooperate with a web server extension 1230 of FIG. 12 to periodically re-execute a query or otherwise update the dynamic data used to populate the web page. For example, a "time until stale" field may be associated with each query included in the spreadsheet web page. When this time expires, end-user network-functionality software 1250 determines whether an appropriate network connection still exists

for updating the dynamic data associated with the query. If the connection still exists, the query is run again, and the dynamic data displayed on the web page is updated with the latest query results. Alternatively, web server extension 1230 may detect that data associated with a registered query in enhanced document web page 1270 has changed and proactively push updated query results to the document for viewing at the client.

In some implementations of this embodiment, the developer constructing embedded application web pages may find that two or more pages at a given Web site (or, for that matter, all pages at a given site) are related and appropriate to be rendered as embedded application web pages. In this case, a spreadsheet version of each of the two or more pages may be created and stored within a single enhanced spreadsheet document, on distinct spreadsheets. These spreadsheets may be linked such that navigating from one to another merely results in the display of a different sheet in the same enhanced spreadsheet document. This capability of storing multiple web pages within a single document provides the ability to improve interoperability and integration between what were originally disconnected pages. It also provides the ability to cache, snapshot, or transmit entire web sites or portions of web sites, and work within an entire web site while disconnected from the hosting network, after having downloaded the single enhanced spreadsheet document containing the multiple embedded application Web pages, and also caching snapshots of any needed population data query results.

As the discussion relating to FIGS. 12-14 illustrates, one embodiment of the present invention enables the development of web pages, web applications and even web sites that can be accessed directly within non-browser client applications, such as word processors, spreadsheet programs, graphics applications and the like. The invention exploits both the power of the productivity application, for content manipulation, calculation capabilities and the like,

and distributed and web-based functionality typically which were previously available only through a separate web browser.

In retrieving network-based content, a user operates a client terminal which is in communication with a server. The server receives a query entered by the user and locates a web-site or the like on the Internet which maintains dynamic data responsive to the query. The query may be stored in one or more repositories. On the client terminal, the repository may be formatted for extensible mark-up language (XML) files. On the server, the repository may exist in a lightweight directory access protocol (LDAP) format. This is because the server may handle requests from a large number of users, thereby necessitating a larger storage capacity more suitable to LDAP or other equivalent structures.

Referring to FIGS. 15A and 15B, therein is depicted an exemplary process 1500 Performed by a user for initiating a query for network-based content (i.e. population data) that can be updated through use of a network-enabling object, and for inserting the population data into a spreadsheet document on the client terminal. Turning to FIG. 15A, the process 1500 begins when a user launches a wizard (step 1502) for initiating a selection of network based content within a spreadsheet productivity application operating on a client terminal. The selection of the wizard may result in the launch of a UI 1620 similar to that depicted in FIG. 16B. The wizard may be launched, for example, by selecting a command 1602 from a menu bar 1604 as shown in FIG. 16A. The command may call visual basic application (VBA) code which initiates the dynamic link library (DLL) object corresponding to the wizard. The user may use the wizard to position and populate a chosen range of cells in a spreadsheet and to select a size of the data to be retrieved and inserted therein.

After the wizard is launched, the user may then select either a stored data query

(i.e. a "pre-canned" query) or may enter a new query (i.e. a "user-selected" query) (step 1504). If a pre-canned query is selected, the process 1500 continues at step 1534 of FIG. 15B, discussed further below. Otherwise, the process 1500 continues to step 1506, where the user may select a source 1610 for the requested data. The source may correspond to a web-site or the like where dynamic network-based data corresponding to a user query is available.

Next, the user may set a maximum data parameter (step 1508), such as a maximum number of rows of data to be entered into a spreadsheet. This may be accomplished by selecting a "maximum row" checkbox (not shown) in the wizard UI and entering a maximum number of desired rows in an accompanying field (not shown). If the user does not choose a parameter for this setting, a default value (i.e. a maximum of 50 rows of data) may be set automatically by default software instructions executed by the wizard. The default value may be changed by an administrator or programmer from an initial setting by a provider of the software wizard.

Continuing to step 1510, the user may then choose whether to select an "advanced mode" option, i.e. by selecting checkbox 1620 of FIG. 16B. The advanced mode option enables the user to specify column names for selected columns in which network-based content will be inserted. The advanced mode may also allow a user to designate whether the query specification is to be saved in a repository on the local client terminal or the server and to preview retrieved data before saving it to a spreadsheet. If advanced mode is not selected, the process 1500 continues to step 1511, described immediately below. Otherwise, the process 1500 continues to step 1517, described further blow.

The user then selects a category and sub-category of data to which the query

pertains (step 1511). Examples of windows for making such selections are shown in FIGS. 16C and 16I. For example, the user may select "Stock Analysis" as the category and further select sub-categories of data such as "current stock price," "price/earning ratio" and the like. These predefined categories and subcategories of data may allow the server to determine a network location for retrieving data corresponding to the query. For example, the server may recognize that a server from BLOOMBERG will contain data for all "Stock Analysis" categories and related sub-categories of data. The correlation between categories and appropriate locations may be stored in a relational database management system (RDBMS) (not shown). A tree structure of categories and sub-categories that are selectable by a user is displayed in FIG. 16C.

After selection of appropriate categories and sub-categories, the user may enter a query (step 1512). The query may be made in a structured query language (SQL) format, as shown in FIG. 16G. Parameter information for the query may be entered as shown in FIG. 16D.

Once formulated, the query is sent to the server, which in turn, retrieves dynamic network-based content corresponding to the query (step 1514). The query may then be saved by the user (step 1516) through a dialog box as shown in FIGS. 16E and 16M, after which, the process 1500 continues to step 1532 below.

Returning to step 1510 above, if the user does select advanced mode, then the process 1500 continues as the follows. The user selects a category and sub-category (if any) (step 1517), in the manner described above for step 1511. Next, at step 1518, the user enters a query in the manner previously described with respect to step 1512.

Since advanced mode has been selected, the user may select column names (step

1520) for columns in which network-based data responsive to the query will be inserted. The user may then preview the data retrieved in response to the query (step 1522). An example of a preview is provided in FIG. 16H.

Next, the software will determine whether multiple functions were selected by the user in the query (step 1524). As used herein, a table refers to data populating more than one row and column of cells. A function is populated in a single field or cell of the spreadsheet. A user may select multiple functions, however, each function must be given a function name for future reference by the wizard. Accordingly, if a single function is selected for the spreadsheet, the process 1500 continues to step 1526, where the software enables a field for allowing the user to store the function name. Such a field is presented in FIG. 16J. If multiple functions are presented, the process 1500 continues to step 1528 where the function field in FIG. 16J is disabled. The function name or query title is then saved (step 1530).

From step 1516 or step 1530 above, the process 1500 continues to step 1532, where the user is prompted to enter input and output views for data requested in the query. The input view may include column headers, static non-network-based data and the like. The output view may be a designation of cells where dynamic network-based content is to be placed. The input view and output view may be selectable view a window such as those displayed in FIGS. 16K and 16L respectively.

At step 1550, the software will determine whether the selected output range is protected. Protection of a cell or range within a spreadsheet is a commonplace spreadsheet function readily known to one of ordinary skill in the art. If the range is protected, the user is prompted to enter a new range (step 1552) and the process 1500 returns to step 1532 above.

Otherwise, the process 1500 continues to step 1554, where the network-based content responsive to the query is placed in the range of the spreadsheet as designated by the user.

The user is next prompted to select whether live updates for the selected data are to be provided (step 1556). Live updating is a feature by which network-based content appearing in the spreadsheet is dynamically updated by the server through use of a network-enabling object in the spreadsheet. The updating may take place at predetermined intervals or upon a change in the data from the web-site or the like from which the data originated. The user may select whether to enable live updating from a checkbox provided in a window, as displayed in FIG. 16J. If live updating is not selected, the user may choose to refresh the data by selecting a refresh button in the open spreadsheet (not shown). The spreadsheet with network-enabled data is then saved (step 1558), after which the process 1500 ends.

Referring now to FIG. 15B, the process 1500 described above continues to step 1534 from step 1504 above when the user chooses a pre-stored or "pre-canned" query. The pre-canned query comes with default preferences including a host server for processing the query and a port for receiving the data over the network. If the user chooses new preferences, the process 1500 continues to step 1538 where the new host and port settings are entered Otherwise, the existing preferences are used by the spreadsheet (step 1536).

Next, the user may select a library in which the pre-canned query is stored. The library may reside on the client terminal, the network server, or some other location accessible over the network. The selection of the library may be made by the user from a window such as is depicted in FIG. 16F.

The software will then enable a display of available queries (step 1544). The user may select one or more of the queries (step 1548) which, in turn, initiates retrieval of the results corresponding to the saved query (step 1548). The process 1500 then continues to step 1550 of FIG. 15A.

Although particular arrangements of display screens and order of steps have been presented and described above with respect to FIGS. 15A-16M, it is to be understood that the data and functionality described therein may be selected, displayed or manipulated in other equivalent manners.

As is made apparent by the examples provided herein, the invention provides for integration of new network enabling functionality into existing productivity applications such that users, as opposed to developers, can easily create and interact with enhanced documents that leverage a variety of distributed services, and that it provides the means for existing service providers to broaden their utility, and overall value through greatly expanded access and availability.

The described architecture for the method and system of the present invention provide a wide array of both client- and server-side benefits. It enhances the functionality available to a user by facilitating connection to network-based services (e.g., Web browsing, conferencing, chat, e-mail, licensing, and workflow), and making these services available within spreadsheet productivity applications. Additionally, because the enhanced productivity application is "network aware," it can provide the end-user with distinct on-line and off-line usage models. For example, the augmented desktop application may, during on-line usage, provide access to network services and, during off-line usage, allow the end-user to work on

previously downloaded content. In addition, data entered by the end-user during off-line usage may be stored for later upload and subsequent processing by a server (i.e., synchronization).

Furthermore, the productivity application with the network-based functionality enhances the UI provided to both developers and users. This enhanced UI may include non-standard toolbars and buttons for launching special-purpose tools and executing commands as described above.

In addition, because the enhanced application can connect to one or more remote servers, the system may efficiently allocate tasks between client and server. For example, a server may be tasked with collecting, processing, and formatting data required by an enhanced application. Enhanced pipes may be established between server and client to facilitate downloading of this information. Both clients and servers may also intelligently pre-fetch content and data in order to respond to anticipated requests by a user. The ability to replicate at least a portion of a Web site within a spreadsheet, and install such a productivity application-based site on a client machine, can improve performance because only real-time dynamic data (i.e., population data) must be obtained over the Internet from the server. Because the present system provides for the separation if desired of wireframe and population data, the wireframe may be treated as a versioned and infrequently downloaded independent element. This decreases bandwidth requirements with use and improves the end-user experience by decreasing screen-refresh times. It also enables more efficient caching of multiple instances at the client because the wireframe is cached only once. Custom, licensed, or public-domain differencing techniques may be employed to exploit this aspect of the present invention. In addition, this separation may be extended to multiple population data blocks per page, each of which can be managed and updated independently.

Furthermore, the present system facilitates optimal workload balancing between client and server. In many cases, it may be desirable to assign various background and agent-type tasks to the server. For example, the server may be programmed to monitor a Web site for changes, and notify the client when a change is detected. Document control (e.g., managing who has permission to modify a document), workflow processes, and productivity application-based conferencing may also be centrally managed by a server. The server may be programmed to detect obsolete versions of runtime code or wireframes on the client and update that code, either automatically or under the end-user's explicit control.

The above architecture facilitates both unstructured use as well as more structured use. Structured use denotes the concept of a packaged application. If one restricts a user's operations, the productivity application has transformed into a new more specialized application, which leverages the desired functionality of the original productivity application. The invention can also be used in unstructured formats, typically in the form of user tools. The user tools will provide quick utilities to enhance the power of the productivity application , such as a screen scraper. Thus, the invention is useful in the format of a highly structured application, quick unstructured utilities, and any type of computer program in between.

Users are also preferably provided with access to, and some control over, runtime utilities via the above architecture. For example, users may initiate conferences and chat sessions with each other from within their augmented desktop-application environments.

Furthermore, since numerous modifications and variations will readily occur to those skilled in the art, it is to be understood that that the present invention is not limited to the exact construction and operation illustrated. Accordingly, all suitable modifications and equivalents are intended to fall within the scope of the appended claims.